

NONLINEAR PARALLEL-IN-TIME SCHUR COMPLEMENT SOLVERS FOR ORDINARY DIFFERENTIAL EQUATIONS

SANTIAGO BADIA[†] AND MARC OLM[‡]

ABSTRACT. In this work, we propose a parallel-in-time solver for linear and nonlinear ordinary differential equations. The approach is based on an efficient multilevel solver of the Schur complement related to a multilevel time partition. For linear problems, the scheme leads to a fast direct method. Next, two different strategies for solving nonlinear ODEs are proposed. First, we consider a Newton method over the global nonlinear ODE, using the multilevel Schur complement solver at every nonlinear iteration. Second, we state the global nonlinear problem in terms of the nonlinear Schur complement (at an arbitrary level), and perform nonlinear iterations over it. Numerical experiments show that the proposed schemes are weakly scalable, i.e., we can efficiently exploit increasing computational resources to solve for more time steps the same problem.

Keywords: Time parallelism, ordinary differential equations, domain decomposition, nonlinear solver, scalability

CONTENTS

1. Introduction	1
2. Statement of the problem	3
3. An ODE direct solver	3
3.1. Application to different time integrators	5
3.2. Parareal interpretation	6
3.3. Parallel efficiency	6
4. Iterative solvers for nonlinear ODEs	8
4.1. Newton-Schur complement methods	8
4.2. Nonlinear Schur complement-Newton methods	8
5. Numerical experiments	10
5.1. Experimental set-up	10
5.2. Two-level solvers	11
5.3. Multilevel solver	13
6. Conclusions	15
References	15

1. INTRODUCTION

At the beginning of the next decade supercomputers are expected to reach a peak performance of one exaflop/s, which implies a 100 times improvement with respect to current supercomputers. This

Date: March 27, 2017.

[†] Universitat Politècnica de Catalunya, Jordi Girona1-3, Edifici C1, E-08034 Barcelona & Centre Internacional de Mètodes Numèrics en Enginyeria, Parc Mediterrani de la Tecnologia, Esteve Terrades 5, E-08860 Castelldefels, Spain E-mail: sbadia@cimne.upc.edu. SB gratefully acknowledges the support received from the Catalan Government through the ICREA Acadèmia Research Program.

[‡] Universitat Politècnica de Catalunya, Jordi Girona1-3, Edifici C1, E-08034 Barcelona & Centre Internacional de Mètodes Numèrics en Enginyeria, Parc Mediterrani de la Tecnologia, Esteve Terrades 5, E-08860 Castelldefels, Spain E-mail: molm@cimne.upc.edu. MO gratefully acknowledges the support from the Agència de Gestió i d'Ajuts Universitaris i de Recerca, under the FI-AGAUR 2015 grant.

improvement will not be based on faster processors, but on a much larger number of processors (in a broad sense). This situation will certainly have an impact in large scale computational science and engineering (CSE). Parallel algorithms will be required to exhibit much higher levels of concurrency, keeping good scalability properties.

When dealing with transient problems, since information always moves forward in time, one can exploit sequentiality. However, the tremendous amounts of parallelism to be exploited in the near future certainly motivates to change this paradigm. One of the motivations to exploit higher levels of parallelism will be to reduce the time-to-solution. In the simulation of ordinary differential equations (ODEs), the way to go is to exploit concurrency in time. The idea is to develop parallel-in-time solvers that provide the solution at all time values in one shot, instead of the traditional sequential approach that exploits the arrow of time. If scalable parallel-in-time solvers are available, the use of higher levels of parallelism will certainly reduce the time-to-solution.

Parallel-in-time solvers are receiving rapidly increasing attention. Different iterative methods have been considered so far, e.g., the *parareal* method [15] or spectral deferred-correction time integrators [10]. With regard to direct methods, time-parallel methods can be found in [12]. In general these methods can exploit low levels of concurrency [8] or are tailored for particular types of equations [13]. We refer to [12] for an excellent up-to-date review of time parallelism. It has also motivated the development of space-time parallel solvers for transient partial differential equations (PDEs) [5, 11].

In this work, we propose a parallel-in-time solver for ODEs that relies on the well-known Schur complement method in linear algebra. For linear (systems of) ODEs, the approach can be understood as a Schur complement solver in time. When the coarse problem is too large compared to the local problems, due to the structure of the coarse problem, we can consider recursively the Schur complement strategy, leading to multilevel implementations, in order to push forward scalability limits. The method can be applied to θ -methods, discontinuous Galerkin (DG) methods, Runge-Kutta methods, and BDF methods. We also note that the proposed method can also be understood as a parareal scheme in which the coarse solver is automatically computed in such a way that the scheme is a direct method (convergence in one iteration is assured). (The interpretation of the parareal method as an approximation of the Schur complement problem has already been pointed out in [11].) As a result, the proposed method solves the drawback of the parareal scheme, i.e., its poor parallel efficiency, inversely proportional to the number of iterations being required by the iterative algorithm. One of the messages of this work is to show that the approximation of the Schur complement in parareal methods does not really pay the price when (just by roughly multiplying by two the number of operations) one can have a highly scalable direct parallel-in-time solver.

In order to extend these ideas to nonlinear PDEs, we consider two different strategies. First, we consider a global linearization of the problem in time using, e.g., Newton's method. We note that the idea of a global linearization of nonlinear ODEs to exploit time-parallelism is not new. It was already considered by Bellen and Zang in 1989 [6]. (In any case, the solvers proposed in [6] are different from the ones presented herein. They are restricted to the Steffensen's linearization method, which leads to a diagonal problem per nonlinear iterations, where parallelization can obviously be used.) After the linearization of the problem, we consider the Schur complement solver commented above at every nonlinear iteration. A second strategy consists in applying the nonlinear Schur complement strategy first, and next to consider the linearization of such operator, leading to nested nonlinear iterations.

The parallel-in-time ideas in this work can naturally be blended with domain decomposition ideas to design highly scalable space-time parallel solvers. We have combined these ideas with a multilevel balancing DD by constraints (BDDC) preconditioner (see [16, 17]) in [5], by judiciously choosing the quantities to be continuous among processors in a space-time partition, i.e. [9]. The resulting space-time parallel solver has been proved to be scalable on thousands of processors for different transient (non)linear PDEs.

The outline of the article is as follows. In Sect. 2, we state the problem. We introduce a time-parallel direct method for linear ODEs based on the computation of a multilevel time Schur complement in Sect. 3. In Sect. 4, we extend the method to nonlinear ODEs, by combining first a Newton linearization step with a Schur complement linear solver and next considering nonlinear Schur complement problems. We

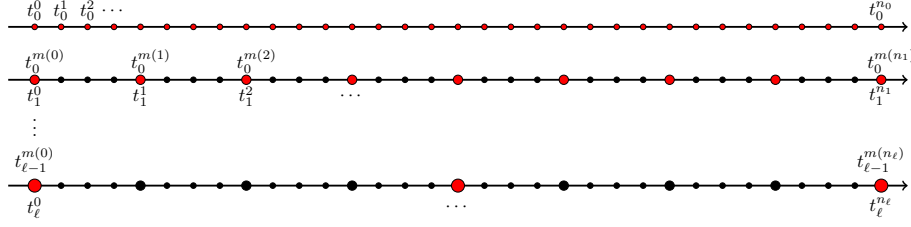


FIGURE 1. Multilevel time partition of $[0, T]$. The subindex in t_α^β denotes the partition level, whereas the superindex denotes the time step in such partition. In order to relate time values of two constitutive levels, we use the notation $t_\alpha^\beta = t_{\alpha-1}^{m(\beta)}$, i.e., the time value t_α^β corresponds to the time step $m(\beta)$ at the previous level.

present a detailed set of numerical experiments in Sect. 5, showing the excellent scalability properties of the proposed methods. Finally, we draw some conclusions in Sect. 6.

2. STATEMENT OF THE PROBLEM

In this section, we develop a parallel direct solver for the numerical approximation of ordinary differential equations (ODEs). We consider a system of ODEs of size m_{unk} :

$$\frac{d\mathbf{u}(t)}{dt} + \kappa(t, \mathbf{u}(t)) = \mathbf{0}, \quad \mathbf{u}(t^0) = \mathbf{u}_0, \quad (1)$$

for $t \in (t^0 = 0, T]$. Let us assume that $\kappa(\cdot, \cdot)$ is continuous with respect to the first argument and Lipschitz continuous with respect to the second argument, and that existence and uniqueness holds.

For the time interval $[0, T]$, we define a hierarchical multilevel partition as follows (see Fig. 1 for a detailed illustration). We define a (level-0) time partition $\{0 = t_0^0, t_0^1, \dots, t_0^{n_0} = T\}$ into n_0 time elements. Next, we consider a (level-1) coarser time partition $\{0 = t_1^0, \dots, t_1^{n_1} = T\}$ into n_1 time subdomains (or level-1 elements), defined by aggregation of elements at the previous level, i.e., for every $i \in \{0, \dots, n_1\}$ there exists an $m(i) \in \{0, \dots, n_0\}$ such that $t_1^i = t_0^{m(i)}$. We proceed recursively, creating coarser partitions for higher levels. We define the time element i at level- k as the time interval (t_k^i, t_k^{i+1}) .

We will present the method in a general way that is independent of the time integration scheme being used. We define the nonlinear operators $\mathcal{A}_0^{i+1} : \mathbf{u}^i \mapsto \mathbf{u}^{i+1}$ for $i \in \{0, \dots, n_0 - 1\}$, such that, given the initial value \mathbf{u}^{i+1} , solves (1) in (t_0^i, t_0^{i+1}) , and provides $\mathbf{u}^{i+1} = \mathbf{u}(t_0^{i+1})$. We conceptually state the solver at the continuous level, even though one can consider different time integration schemes instead, e.g., θ -methods, DG methods, or Runge-Kutta methods. The use of BDF-type schemes requires some further elaboration. We refer to Sect. 3.1 for more details.

3. AN ODE DIRECT SOLVER

In this section, we assume that $\kappa(t, \cdot)$ is a linear operator. (The nonlinear extension is described in Sect. 4.) In this case, it is easy to check that $\mathcal{A}_0^{i+1}(\cdot)$ is an affine mapping, and we have $\mathcal{A}_0^{i+1}(\mathbf{u}^i) = \Phi_0^{i+1}\mathbf{u}^i + \mathbf{g}_0^{i+1}$, where Φ_0^{i+1} is a linear operator (an $m_{\text{unk}} \times m_{\text{unk}}$ matrix). Both Φ_0^{i+1} and \mathbf{g}_0^{i+1} can be explicitly computed from $\kappa(\cdot, \cdot)$ and \mathbf{f} for a given time integration scheme. Thus, the global problem (1) for linear ODEs can be stated in algebraic form as:

$$\begin{pmatrix} \mathcal{I} & & & \\ -\Phi_0^1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\Phi_0^{n_0} & \mathcal{I} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0^0 \\ \mathbf{u}_0^1 \\ \vdots \\ \mathbf{u}_0^{n_0} \end{pmatrix} = \begin{pmatrix} \mathbf{u}_0 \\ \mathbf{g}_0^1 \\ \vdots \\ \mathbf{g}_0^{n_0} \end{pmatrix}, \quad (2)$$

where \mathcal{I} is the identity matrix (of size $m_{\text{unk}} \times m_{\text{unk}}$). We can also represent the global system (2) in compact notation with

$$\mathbf{K}_0 \mathbf{u}_0 = \mathbf{g}_0, \quad \text{or equivalently} \quad \begin{pmatrix} \mathbf{K}_0^{II} & \mathbf{K}_0^{I\Gamma} \\ \mathbf{K}_0^{\Gamma I} & \mathbf{K}_0^{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0^I \\ \mathbf{u}_0^\Gamma \end{pmatrix} = \begin{pmatrix} \mathbf{g}_0^I \\ \mathbf{g}_0^\Gamma \end{pmatrix},$$

where we have considered a segregation of degrees of freedom (DOFs) at level-0 \mathbf{u}_0 into the *interface* DOFs \mathbf{u}_0^Γ , namely the time step values that are also in the level-1 partition, and the *interior* dofs \mathbf{u}_0^I . \mathbf{K}_0 is a 2-banded lower block-triangular matrix. In order to define the Schur complement problem, we first consider an *interior correction* of the problem at hand,

$$\mathbf{K}_0^{II} \mathbf{v}_0^I = \mathbf{g}_0^I, \quad (3)$$

i.e., we solve the system (2) in the subspace of vectors that vanish in the level-1 time values $t_1^0, \dots, t_1^{n_1}$. The solution of (3) involves n_1 independent local ODE problems: solve for $i = 0, \dots, n_1 - 1$

$$\begin{pmatrix} \mathcal{I} & & & \\ -\Phi_0^{m(i)+1} & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\Phi_0^{m(i+1)-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} \mathbf{v}_0^{m(i)} \\ \mathbf{v}_0^{m(i)+1} \\ \vdots \\ \mathbf{v}_0^{m(i+1)-1} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{g}_0^{m(i)+1} \\ \vdots \\ \mathbf{g}_0^{m(i+1)-1} \end{pmatrix}. \quad (4)$$

After the interior correction, we must solve the problem

$$\begin{pmatrix} \mathbf{K}_0^{II} & \mathbf{K}_0^{I\Gamma} \\ \mathbf{K}_0^{\Gamma I} & \mathbf{K}_0^{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} \delta \mathbf{u}_0^I \\ \mathbf{u}_0^\Gamma \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{g}_0^\Gamma \end{pmatrix}, \quad \text{and compute} \quad \mathbf{u}_0^I = \mathbf{v}_0^I + \delta \mathbf{u}_0^I. \quad (5)$$

In order to solve (5), we define the following extension operator (usually denoted as the harmonic extension operator in the frame of domain decomposition solvers for PDEs):

$$\begin{pmatrix} \mathbf{K}_0^{II} & \mathbf{K}_0^{I\Gamma} \\ \mathbf{0} & \mathbf{I}_\Gamma \end{pmatrix} \begin{pmatrix} \mathbf{E}_0^I \\ \mathbf{E}_0^\Gamma \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{I}_\Gamma \end{pmatrix}, \quad \text{thus} \quad \mathbf{E}_0 = \begin{pmatrix} -(\mathbf{K}_0^{II})^{-1} \mathbf{K}_0^{I\Gamma} \\ \mathbf{I}_\Gamma \end{pmatrix}. \quad (6)$$

Thus, using the fact that $\mathbf{u}_0^\Gamma = \mathbf{u}_1$, the Schur complement of level-0 reads:

$$(\mathbf{K}_0^{\Gamma\Gamma} + \mathbf{K}_0^{I\Gamma} \mathbf{E}_0) \mathbf{u}_1 = \mathbf{g}_0^\Gamma - \mathbf{K}_0^{I\Gamma} \mathbf{v}_0^I, \quad \text{represented by} \quad \mathbf{K}_1 \mathbf{u}_1 = \mathbf{g}_1. \quad (7)$$

The Schur complement of the level-0 system is the level-1 problem. By construction, the extension operator solution of (6) can be written as a block-diagonal matrix $\mathbf{E}_0 = \text{diag}(\mathbf{e}_{(0)}, \mathbf{e}_{(1)}, \dots, \mathbf{e}_{(n_1-1)}, \mathcal{I})$, which involves $n_1 \times m_{\text{unk}}$ independent local ODE problems: solve for $i = 0, \dots, n_1 - 1$

$$\begin{pmatrix} \mathcal{I} & & & \\ -\Phi_0^{m(i)+1} & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\Phi_0^{m(i+1)-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} (\mathbf{e}_{(i)})_0^{m(i)} \\ (\mathbf{e}_{(i)})_0^{m(i)+1} \\ \vdots \\ (\mathbf{e}_{(i)})_0^{m(i+1)-1} \end{pmatrix} = \begin{pmatrix} \mathcal{I} \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \quad (8)$$

After some manipulation, the Schur complement problem (5) can be written as:

$$\begin{pmatrix} \mathcal{I} & & & \\ -\Phi_1^1 & \mathcal{I} & & \\ & \ddots & \ddots & \\ & & -\Phi_1^{n_1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1^0 \\ \mathbf{u}_1^1 \\ \vdots \\ \mathbf{u}_1^{n_1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}_0 \\ \mathbf{g}_1^{m(1)} \\ \vdots \\ \mathbf{g}_1^{m(n_1)} \end{pmatrix}, \quad (9)$$

where $\Phi_1^i \doteq \Phi_0^{m(i)} \mathbf{e}_{(i-1)}^{m(i)-1}$ and $\mathbf{g}_1^i \doteq \mathbf{g}_0^{m(i)} + \Phi_0^{m(i)-1} \mathbf{v}_0^{m(i)-1}$. Thus, the Schur complement matrix \mathbf{K}_1 at the next level has also the same structure as the original problem, i.e., it is a ODE-type solver over the coarser level-1 time partition. The computation of the interior correction, the extension operator \mathbf{E}_0 , the Schur complement matrix \mathbf{K}_1 , and the Schur complement right-hand side \mathbf{g}_1 can readily be computed in parallel. In Alg. 1 (for $k = 0$), all the steps to assemble the Schur complement problem are listed, indicating on the left-hand side the line task corresponding level.

Algorithm 1: Schur complement set-up (level- k)

-
- Data:** $\mathbf{K}_k, \mathbf{g}_k$
Result: $\mathbf{v}_k, \mathbf{E}_k, \mathbf{K}_{k+1}, \mathbf{g}_{k+1}$
- 1: Compute the interior correction \mathbf{v}_k solution of (3), by solving the n_{k+1} local ODE problems (4) for
 $i = 0, \dots, n_{k+1} - 1$ $k - 1$
 - 2: Compute the extension operator \mathbf{E}_k solution of (6), by solving the $n_{k+1} \times m_{\text{unk}}$ problems (8) for
 $i = 0, \dots, n_{k+1} - 1$, and compute $\mathbf{K}_k^{I\Gamma} \mathbf{E}_0^I$ and $-\mathbf{K}_k^{I\Gamma} \mathbf{u}_k^I$ in (7) $k - 1$
 - 3: Assemble the Schur complement system (9), i.e., \mathbf{K}_{k+1} and \mathbf{g}_{k+1} (see (7)) $k - 1 \rightarrow k$
-

In a two-level implementation, i.e., $\ell = 1$, the Schur complement problem (5) would be computed in serial. It would finally lead to $\mathbf{u}_0 = \mathbf{v}_0 + \mathbf{E}_0 \mathbf{u}_1$. This approach leads to n_1 independent level-0 problems of size $\frac{n_0}{n_1}$ and one coarse level-1 problem of size n_1 . Clearly, when n_1 increases, the coarse problem becomes the bottleneck of the simulations. In order to push forward the scalability limits of this approach, we can consider a multilevel Schur complement technique. When n_1 exceeds $\frac{n_0}{n_1}$, since (9) has the same structure as the original system (2), one can consider the same Schur complement approach for the level-1 system, leading to a three-level algorithm. We can proceed recursively to include an arbitrary number of levels. In Alg. 2, we state the multilevel Schur complement ODE solver. We comment on the parallel efficiency and computational cost of this algorithm in Sect. 3.3.

Algorithm 2: Multilevel Schur complement ODE solver

-
- Data:** $\mathbf{K}_0, \mathbf{g}_0$
Result: $\mathbf{u}_0 = \mathbf{K}_0^{-1} \mathbf{g}_0$
- 1: **for** $k = 0, \dots, \ell - 1$ **do**
 - 2: | Call Alg. 1 with $(\mathbf{K}_k, \mathbf{g}_k)$ to get $(\mathbf{v}_k, \mathbf{E}_k, \mathbf{K}_{k+1}, \mathbf{g}_{k+1})$ $k, k + 1$
 - 3: **end**
 - 4: Solve $\mathbf{K}_\ell \mathbf{u}_\ell = \mathbf{g}_\ell$ ℓ
 - 5: **for** $k = \ell - 1, \dots, 0$ **do**
 - 6: | Compute $\mathbf{u}_k = \mathbf{v}_k + \mathbf{E}_k \mathbf{u}_{k+1}$ k
 - 7: **end**
-

3.1. Application to different time integrators. The previous approach can straightforwardly be used for θ -methods. For DG and Runge-Kutta methods, we can require multiple intermediate stages to move from t_0^i to t_0^{i+1} . In the DG case, we have some additional time values per time element. We can consider that all the element values but the last one are eliminated at the element level, using the so-called static condensation technique. In this case, the resulting discrete problem can be stated as in 2. The DG method being used only affects the expression of Φ_0^{i+1} and \mathbf{g}_0^{i+1} . (We note that DG methods do not satisfy time causality at the element level, but it does not affect the lower 2-banded block-triangular structure after the elimination of “interior” element values.) We proceed analogously for multi-stage Runge-Kutta methods.

BDF schemes (of second and higher order) slightly differ from the fact that the computation of the solution at a given time step not only requires value from the previous time step, but some additional stages. For a BDF(X) scheme, the system matrix in (2) is a (X+1)-banded lower block-triangular matrix. It affects the concept of interface nodes Γ ; to decouple the global problems into local problems, we require to increase the size of the interface X times. As a result, the coarse-scale space dimension is X times larger, as well as the number of coarse space basis functions being computed. In this sense, high order BDF schemes are a bad choice when dealing with parallel computations, since the interface among subdomains increases, with the corresponding computational cost, due to a loss of locality with respect to the continuous problem. High order Runge-Kutta or DG methods are

better suited for time-parallel computations. In any case, after considering the modification described above, the techniques proposed in this work can be applied to BDF methods.

3.2. Parareal interpretation. The multilevel Schur complement solver defined in Alg. 2 can also be understood as a (multilevel) parareal scheme. In the parareal scheme, we consider a coarse solver to provide initial conditions to local fine solvers. Instead, in the Schur complement method, one first computes the (fine) interior correction, which is required to the assembly of the right-hand side in the coarse solver. The method above can be stated in a different way, by defining the restriction operator \mathbf{F}_0 as follows:

$$\begin{pmatrix} \mathbf{F}_0^I & \mathbf{F}_0^\Gamma \end{pmatrix} \begin{pmatrix} \mathbf{K}_0^{II} & 0 \\ \mathbf{K}_0^{\Gamma I} & \mathbf{I}_\Gamma \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{I}_\Gamma \end{pmatrix}, \quad \text{thus} \quad \mathbf{F}_0 = \begin{pmatrix} -\mathbf{K}_0^{\Gamma I}(\mathbf{K}_0^{II})^{-1} & \mathbf{I}_\Gamma \end{pmatrix}.$$

Analogously to the extension operator \mathbf{E}_0 , the computation of the restriction operator is a block-diagonal matrix $\mathbf{F}_0 = \text{diag}(\mathcal{I}, \mathbf{f}_{(0)}^T, \mathbf{f}_{(1)}^T, \dots, \mathbf{f}_{(n_1-1)}^T)$, which involves $n_1 \times m_{\text{unk}}$ independent local (backwards) ODE problems: solve for $i = 0, \dots, n_1 - 1$

$$\begin{pmatrix} \mathcal{I} & -(\Phi_0^{m(i)+2})^T & & \\ & \mathcal{I} & \ddots & \\ & & \ddots & -(\Phi_0^{m(i+1)})^T \\ & & & \mathcal{I} \end{pmatrix} \begin{pmatrix} \mathbf{f}_{(i)}^{m(i)+1} \\ \mathbf{f}_{(i)}^{m(i)+2} \\ \vdots \\ \mathbf{f}_{(i)}^{m(i+1)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \mathcal{I} \end{pmatrix}.$$

The well-posedness of the backward problem is a direct consequence of the well-posedness of its transpose, the forward problem. Thus, the Schur complement problem reads:

$$\mathbf{K}_1 \mathbf{u}_1 = \mathbf{g}_0^\Gamma + \mathbf{F}_0^I \mathbf{g}_0^I = \mathbf{g}_1.$$

The coarse parareal problem in the Schur complement method is of Petrov-Galerkin type, using as coarse trial space the range of \mathbf{E}_0 and as coarse test space the range of \mathbf{F}_0^T , i.e.,

$$\mathbf{K}_1 = \mathbf{F}_0 \mathbf{K}_0 \mathbf{E}_0, \quad \mathbf{g}_1 = \mathbf{F}_0 \mathbf{g}_0. \quad (10)$$

It is easy to check that (5) and (10) are equivalent. The fine solver is simply the interior correction (3). (We note that fine and coarse corrections are independent, since they are \mathbf{K}_0 -orthogonal.). The definition of the coarse space is automatic and the method is not an iterative but a direct solver. As a result, the method does not suffer from the low parallel efficiency of parareal methods, which is proportional to the inverse of parareal iterations [15]. Even though the implementation that involves the computation of the trial and test coarse spaces is the one being used in non-symmetric PDE solvers with inexact Schur complement preconditioning (see, e.g., [1]), it is not convenient for ODE solvers, since it involves an additional fine solver.

Fig. 2 depicts forward and backward coarse functions for a test problem. The local oscillations in the DG(1) and DG(2) schemes are due to the fact that time causality does not hold inside the element. (We note that the previous developments have been considered after eliminating (using element-wise solvers) all the element values but the last one (in time).) Fig. 3 shows coarse level-1, fine level-0, and full solution for a selected simple problem with different coarse DOFs position consideration.

3.3. Parallel efficiency. Let us consider the multilevel Schur complement solver in Alg. 2 for a linear ODE. Using the same approach as in multigrid methods, we can consider a fixed coarsening ratio θ between subdomains, and leave free the number of levels ℓ required for a particular simulation with n_0 time step values. The number of processors being used is assumed to be equal to $n_1 = n_0/\theta$, i.e., the number of subdomains at level-1, and the number of time steps per processor at all levels is θ (at the last level it can be smaller). Thus, we define $\ell + 1 = \text{ceiling}(\frac{\log n_0}{\log \theta})$, where $\text{ceiling}(A)$ returns the least integer greater than or equal to A .

Alg. 2 requires at levels $0, \dots, \ell - 1$ to solve $1 + m_{\text{unk}}$ local linear ODE problems with θ time steps per processor (see Alg. 1), one to compute the interior correction and m_{unk} to compute the extension

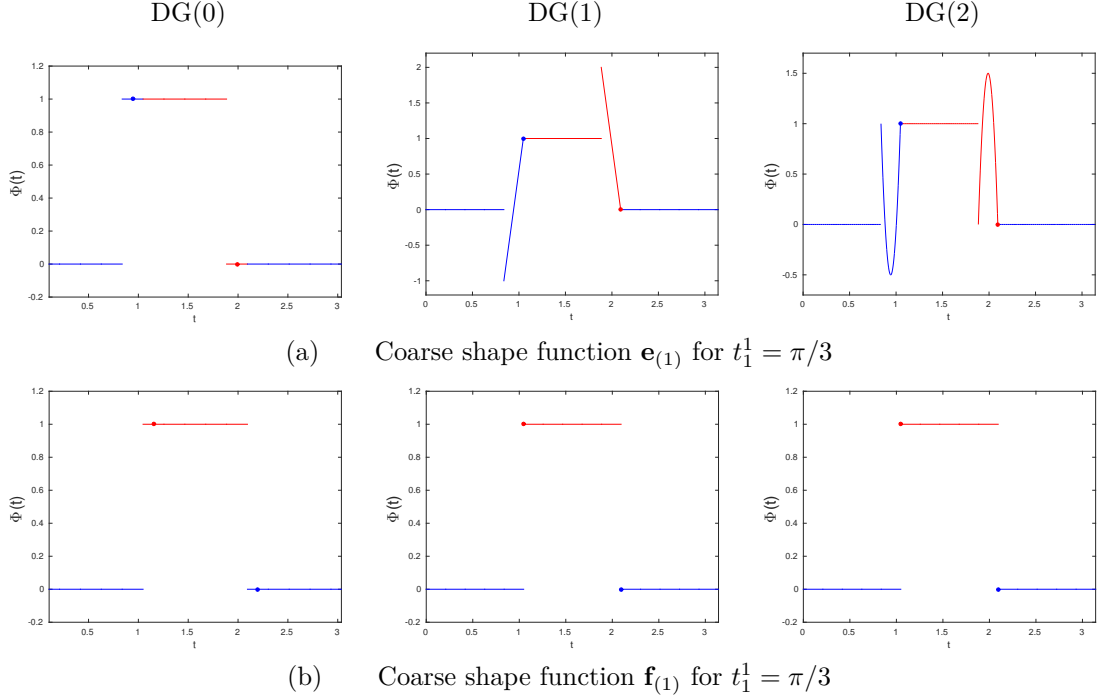


FIGURE 2. Coarse shape functions $\mathbf{e}_{(1)}$ and $\mathbf{f}_{(1)}$ ($t_1^1 = \pi/3$) for the simple transport operator $\frac{du}{dt}$. A partition of an interval $[0, \pi]$ into three subdomains is considered. Each subdomain is partitioned into 5 time elements. Sub-intervals are depicted in consecutive different colour in order to aid visualization, and dots aid to identify coarse DOFs position (in the center of the element for DG(0)).

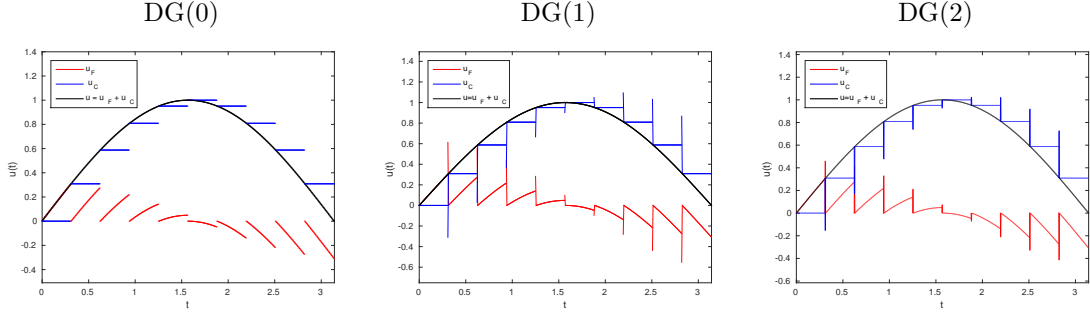


FIGURE 3. Decomposition of \mathbf{u}_0 into fine $\mathbf{E}_0 \mathbf{u}_0$ and coarse \mathbf{u}_1 component for the simple problem $\partial_t u = \cos(t)$, on $[0, \pi]$. The time domain is partitioned into 10 subdomains and each subdomain is an aggregation of 50 elements.

operator. One linear ODE with at most θ time steps must be solved at the last level. The different levels must be computed in a sequential way.

Based on the solves described above, we can estimate the order of floating-point operations (FLOPs) required to solve the linear ODE. Sequentially, it is of the order of $\text{FLOP}_0 \approx n_0(m_{\text{unk}}^2 + m_{\text{unk}})$, since we require $m_{\text{unk}}^2 + m_{\text{unk}}$ operations per time step. On the other hand, the number of FLOPs required to compute the same problem in parallel using Alg. 2 is the one needed to solve $(1 + m_{\text{unk}})$ problems of size $n_0, n_0/\theta, \dots, n_0/\theta^\ell$, thus $\text{FLOP}_p \approx \text{FLOP}_0(1 + m_{\text{unk}})(1 - \frac{1}{\theta})^{-1}$, but these operations can

be performed in parallel exploiting distributed memory machines. With regard to time, the total CPU time of the multilevel Schur complement in Alg. 2 is the aggregation of the CPU time of the solution of $(1 + m_{\text{unk}})$ linear ODEs with θ time steps for all levels. Thus, the parallel CPU time is $\text{CPU}_p \approx \ell\theta(m_{\text{unk}}^2 + m_{\text{unk}})(1 + m_{\text{unk}})$, whereas the serial CPU time is $\text{CPU}_0 \approx n_0(m_{\text{unk}}^2 + m_{\text{unk}})$. As a result, the CPU time linearly depends on the size of the global system in a very mild logarithmic way, i.e., it increases with $\log n_0$ due to the expression of ℓ , and the parallel algorithm will rapidly lead to a shorter time-to-solution than the serial solver.¹ As a result, the speed-up of the proposed direct solver is $S = \text{CPU}_0/\text{CPU}_p \approx \text{Pl}^{-1}(1 + m_{\text{unk}})^{-1}$. The speed-up is quasi-linear (it only increases with $\log n_0$), and thus algorithmically strongly scalable. Analogously, the method is algorithmically weakly scalable, because the total CPU time does not depend on the number of processors or global system size (apart from the logarithmic term).

4. ITERATIVE SOLVERS FOR NONLINEAR ODES

In this section, we assume that $\kappa(\cdot, \cdot)$ is nonlinear. The nonlinear ODE (1) in $(0, T]$ can be stated in compact form as $\mathcal{A}_0(\mathbf{u}_0) = \mathbf{0}$, which can also be split into interior and interface time steps as follows:

$$\begin{pmatrix} \mathcal{A}_0^I(\mathbf{u}_0) \\ \mathcal{A}_0^{\Gamma}(\mathbf{u}_0) \end{pmatrix} = \mathbf{0}. \quad (11)$$

We consider two different types of solvers for the nonlinear problem.

4.1. Newton-Schur complement methods. In order to solve the nonlinear ODE, we can use Newton's method over the global-in-time problem, and solve at every iteration a linear ODE using the multilevel Schur complement in Alg. 2. We can compute the Jacobian matrix related to (11) around a point $\bar{\mathbf{u}}_0$ as:

$$\mathcal{J}_0(\bar{\mathbf{u}}_0) \doteq \frac{\partial \mathcal{A}_0}{\partial \mathbf{u}_0}(\bar{\mathbf{u}}_0) = \frac{d\bar{\mathbf{u}}_0}{dt} + \frac{\partial \mathcal{K}_0}{\partial \mathbf{u}_0}(\bar{\mathbf{u}}_0), \quad (12)$$

where we have used the fact that \mathcal{A}_0 has two terms, one related to the time derivative and the other one related to $\kappa(t, \cdot)$ (see 1). Thus, to solve (12) involves the solution of a linear ODE of the form (2).

The resulting multilevel Newton-Schur complement solver for nonlinear ODEs is stated in Alg. 3. Figure 4 shows the solution iterates using this algorithm for a selected nonlinear problem with known analytic solution $u = \sin(t)$. Each solution update obtained from a linearized problem is solved with the direct solver in Alg. 2 using two levels, i.e., $\ell = 1$. The nonlinear iterations of the Newton-Schur complement methods do not depend on the partition being used, since it is a linearization of the global problem and a (parallel) direct solver is used at every nonlinear iteration. In any case, the comparison against the sequential approach is more complicated here, since different nonlinear iterations (local vs. global) are being used in every case. We note that we can readily consider other iterative methods, e.g., Picard's method or Anderson acceleration techniques. In the case of Picard's method, the computation of the Jacobian is not required. Instead, the operator \mathcal{A}_0 must be written as $\mathcal{A}_0(\mathbf{u}_0) = \tilde{\mathcal{A}}_0(\mathbf{u}_0)\mathbf{u}_0$ and use at every nonlinear iteration the linear operator $\tilde{\mathcal{A}}_0(\bar{\mathbf{u}}_0)$ instead of $\mathcal{J}_0(\bar{\mathbf{u}}_0)$. In Sect. 5, we consider a hybrid Picard-Newton nonlinear solver.

4.2. Nonlinear Schur complement-Newton methods. Following the ideas in nonlinear domain decomposition (see, e.g., [7, 14]), one can state the problem as a nonlinear Schur complement at a given level and next its linearization using, e.g., Newton's method. In order to present the problem, we consider the two-level case. Later, the algorithm will be extended to multiple levels. Let us define the level-1 nonlinear Schur complement problem

$$\mathcal{A}_1(\mathbf{u}_1) = \mathcal{A}_0^{\Gamma}(\mathcal{E}_0(\mathbf{u}_1)), \quad \text{where} \quad \mathcal{E}_0(\mathbf{u}_1) \doteq [\mathcal{E}_0^I(\mathbf{u}_1), \mathbf{u}_1]^T,$$

¹We note that the communications are not considered in the previous estimations. In any case, it has been experimentally observed that the communication time in (incomplete) multilevel Schur complement methods is small compared to the computation time up to almost half a million tasks in [4].

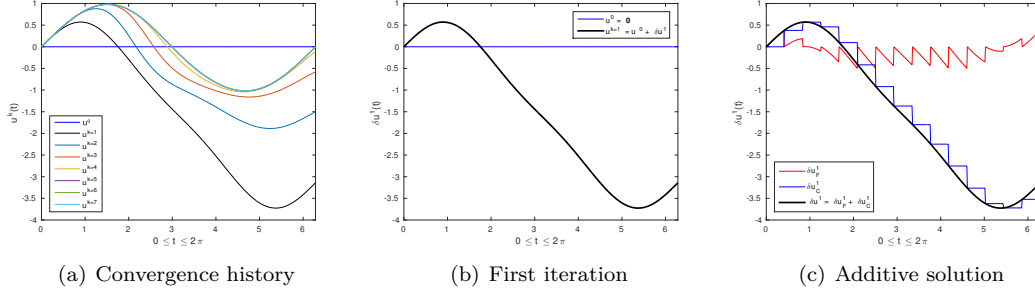


FIGURE 4. Iterations for the solution of the nonlinear equation $\partial_t u - u^2 = \cos(t) - \sin^2(t)$ on $t = [0, 2\pi]$ using Newton's method and the parallel ODE direct solver. Fine and coarse solutions for the first nonlinear solution update δu^1 . The time interval is discretized with 500 time steps divided into 15 subdomains. DG(0) (equivalent to Backward-Euler) is used.

Algorithm 3: Newton-Schur complement solver

Data: \mathbf{u}^0
Result: \mathbf{u}_0 such that $\mathcal{A}_0(\mathbf{u}_0) = 0$

- 1: $\mathbf{z} \leftarrow \mathbf{u}^0$ % Initial guess k
- 2: **while** not convergence **do**
- 3: Solve problem $\mathcal{J}_0(\mathbf{z})\mathbf{y} = -\mathcal{A}_0(\mathbf{z})$ using the multilevel Schur complement Alg. 2 with $0, \dots, \ell$
 $(\mathcal{J}_0(\mathbf{z}), -\mathcal{A}_0(\mathbf{z}))$ k
- 4: Assign $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{y}$
- 5: **end**
- 6: Return \mathbf{z} k, \dots, ℓ

is the nonlinear harmonic extension operator, solution of

$$\mathcal{A}_0^I(\mathcal{E}_0(\mathbf{u}_1)) \doteq \mathcal{A}_0^I([\mathcal{E}_0^I(\mathbf{u}_1), \mathbf{u}_1]^T) = \mathbf{0}. \quad (13)$$

We note that the computation of $\mathcal{E}_0(\mathbf{u}_1)$ involves n_1 independent local nonlinear ODE solvers, using the same rationale as for the linear case. We denote by $[i]$ the time steps at level-0 in the time interval (t_1^i, t_1^{i+1}) , i.e., $t_0^{m(i-1)+1}, \dots, t_0^{m(i)-1}$. Thus, (13) can be computed as:

$$\mathcal{A}_0^{[i]}(\mathcal{E}_0^{[i]}(\mathbf{u}_1^i)) = \mathbf{0}, \quad \text{for } i = 0, \dots, n_1 - 1. \quad (14)$$

Next, we apply linearization over the level-1 nonlinear Schur complement problem, e.g., Newton's method. In this case, using the implicit function theorem (see [14]), we know that

$$\begin{aligned} \mathcal{J}_1(\bar{\mathbf{u}}_1) &\doteq \frac{\partial \mathcal{A}_1}{\partial \mathbf{u}_1}(\bar{\mathbf{u}}_1) \\ &= \mathcal{J}_0^{\Gamma\Gamma}(\mathcal{E}_0(\bar{\mathbf{u}}_1)) - \mathcal{J}_0^{\Gamma\Gamma}(\mathcal{E}_0(\bar{\mathbf{u}}_1))\mathcal{J}_0^{II}(\mathcal{E}_0(\bar{\mathbf{u}}_1))^{-1}\mathcal{J}_0^{\Gamma\Gamma}(\mathcal{E}_0(\bar{\mathbf{u}}_1)), \end{aligned} \quad (15)$$

i.e., the level-1 Schur complement matrix of $\mathcal{J}_0(\mathcal{E}_0(\bar{\mathbf{u}}_1))$. The computation of the Schur complement operator can be done in parallel, as described in Sect. 3. As a result, the only difference between the Newton-Schur complement Alg. 5 and the nonlinear Schur complement-Newton Alg. 4 (for $k = 1$) is the nonlinear interior correction being computed in the second case (stated in Alg. 4). Using recursion, we can extend the nonlinear Schur complement-Newton algorithm to arbitrary levels. In Alg. 5 we state the algorithm to solve the nonlinear Schur complement at level- k using Newton's method for both the nonlinear extensions in Alg. 4 and the problem itself. Thus, the resulting method involves nested Newton iterations. Let us describe these algorithms in detail.

We first consider the nonlinear harmonic extension at level- k . Since the values at the previous level are known, i.e., \mathbf{u}_k^Γ is fixed, the computation of $\mathcal{A}_k(\mathbf{u}_k) = \mathbf{0}$ is to be computed solving local nonlinear ODE problems (14) (at level- k). Alg. 4 states the computation of such nonlinear ODE problem for a given level k and level- k time element i , i.e., (t_k^i, t_k^{i+1}) . We can solve the local nonlinear ODE using Newton's method. In any case, unless $k = 0$, we do not have an explicit expression of the nonlinear ODE. Thus, in order to compute the Jacobian (Eq. (15) for level k), we require to compute the $k - 1$ level extension in i . It leads to the deployment of $\text{card}([i])$ nonlinear ODEs at the level $k - 1$ in elements $j \in [i]$. Again, if the next level is not the level-0, we have to proceed recursively to solve these local problems. All these tasks are described in Alg. 4.

Algorithm 4: Nonlinear harmonic extension

Data: $k, i, \mathbf{v}, \mathbf{v}_0$
Result: $\mathcal{E}_k^{[i]}(\mathbf{v}), \mathcal{J}_k^{[i]}(\mathcal{E}_k^{[i]}(\mathbf{v})), -\mathcal{A}_k^{[i]}(\mathcal{E}_k^{[i]}(\mathbf{v}))$

```

1: if  $k = 0$  then
2:   Solve  $\mathcal{A}_0^{[i]}(\mathcal{E}_0^{[i]}(\mathbf{v})) = \mathbf{0}$  using Newton's method (local nonlinear ODE), in order to get  $\mathcal{E}_0^{[i]}(\mathbf{v})$       0
3:   Compute  $\mathcal{J}_0^{[i]}(\mathcal{E}_0^{[i]}(\mathbf{v})), -\mathcal{A}_0^{[i]}(\mathcal{E}_0^{[i]}(\mathbf{v}))$       0
4: else
5:   % Solve  $\mathcal{A}_k^{[i]}(\mathcal{E}_k^{[i]}(\mathbf{v})) = \mathbf{0}$  using Newton's method as follows
6:    $\mathbf{z}^{[i]} = \mathbf{v}_0^{[i]}$  % Initial guess       $k$ 
7:   while not convergence do
8:     for  $j \in [i]$  do
9:       Solve  $\mathcal{A}_{k-1}^{[j]}(\mathcal{E}_{k-1}^{[j]}(\mathbf{z}^j)) = \mathbf{0}$  using Alg. 4 with  $(k - 1, j, \mathbf{z}^j, \mathbf{v}_0)$        $k - 1$ 
10:      Compute  $\mathcal{J}_{k-1}^{[j]}(\mathcal{E}_{k-1}^{[j]}(\mathbf{z}^j))$  and  $-\mathcal{A}_{k-1}^{[j]}(\mathcal{E}_{k-1}^{[j]}(\mathbf{z}^j))$        $k - 1$ 
11:    end
12:    Assemble  $\mathcal{J}_k^{[i]}(\mathbf{z})$  and  $-\mathcal{A}_k^{[i]}(\mathbf{z})$  using the local contributions in 1.10 and formula (15)       $k - 1 \rightarrow k$ 
13:    Solve  $\mathcal{J}_k^{[i]}(\mathbf{z})\mathbf{y} = -\mathcal{A}_k^{[i]}(\mathbf{z})$  and assign  $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{y}$        $k$ 
14:  end
15:  Return  $(\mathbf{z}, \mathcal{J}_k^{[i]}(\mathbf{z}), -\mathcal{A}_k^{[i]}(\mathbf{z}))$        $k$ 
16: end

```

Once we have defined the nonlinear extension operator, we can state the level- k nonlinear Schur complement solved with Newton's method. Using the expression of the Jacobian in (15) (for level k), we require to compute the nonlinear extension at the next level, using Alg. 4 at all level- k elements. It leads to a level- k linear ODE to be solved, for which we can use the multilevel Schur complement approach in Alg. 2, exploiting levels k, \dots, ℓ in its solution. As commented above, the level-1 nonlinear Schur complement consists in Newton iterations like in Alg. (3), but with the main difference that one performs a nonlinear interior correction of the values at level-0, solving the nonlinear ODE locally. The level- k nonlinear Schur complement requires nested nonlinear iterations to perform the nonlinear harmonic extension at that level, but the Jacobian problem to be solved only involves levels k, \dots, ℓ . As commented above, other linearization techniques can readily be used.

5. NUMERICAL EXPERIMENTS

5.1. Experimental set-up. In this section we evaluate the weak scalability of the proposed methods. We consider the time interval $(0, T]$, which is divided into n_0 time elements. The parallel solver relies on a level-1 coarser time partition into n_1 time elements (time subdomains); thus, every level-1 time element is defined by aggregation of $\frac{n_0}{n_1}$ level-0 time elements. n_1 processors are used for the simulations in all cases. One higher level of coarsening gives a partition into n_2 elements; analogously, every level-2 time element is defined by aggregation of $\frac{n_1}{n_2}$ level-1 time elements. See Fig. 1 for a graphical illustration. Only a subset of n_2 processors have duties at level-2. Two-level methods involve level-0 and level-1 duties, whereas the three-level methods also involve level-2 duties. Higher levels have not

Algorithm 5: level- k nonlinear Schur complement-Newton solver

Data: $k, \mathbf{u}^0, \mathbf{u}_0^0$
Result: \mathbf{u}_0 such that $\mathcal{A}_0(\mathbf{u}_0) = 0$

```

1:  $\mathbf{z}_0 = \mathbf{u}_0^0$  % Initial guess  $k$ 
2: while not convergence do
3:   for  $i = 0, \dots, n_k$  do
4:     Compute nonlinear harmonic extension  $\mathcal{E}_{k-1}^{[i]}(\mathbf{z}_k^i)$  and the local contributions  $\mathcal{J}_{k-1}^{[i]}(\mathcal{E}_{k-1}^{[i]}(\mathbf{z}_k^i)),$ 
        $-\mathcal{A}_{k-1}^{[i]}(\mathcal{E}_{k-1}^{[i]}(\mathbf{z}_k^i))$  using Alg. 4 with  $(k-1, i, \mathbf{z}_k^i, \mathbf{z})$   $k-1, \dots, 0$ 
5:   end
6:   Assemble the level- $k$  Schur complement around  $\mathcal{E}_{k-1}(\mathbf{z}_k)$  using the local contributions in 1.4 and
       formula (15)  $k-1 \rightarrow k$ 
7:   Solve problem  $\mathcal{J}_k(\mathbf{z}_k)\mathbf{y} = -\mathcal{A}_k(\mathbf{z}_k)$  using the multilevel Schur complement Alg. 2 with
        $(\mathcal{J}_k(\mathbf{z}_k), -\mathcal{A}_k(\mathbf{z}_k))$   $k, \dots, \ell$ 
8:   Assign  $\mathbf{z}_k \leftarrow \mathbf{z}_k + \mathbf{y}$   $k$ 
9: end
10: Return  $\mathbf{z}_0$   $k, \dots, \ell$ 

```

been needed at the scales considered below but could be needed at largest scales to keep good weak scalability.

Three different approaches are considered for solving the Lotka-Volterra system of nonlinear ODEs: 1) the pure sequential approach (no parallelism is exploited, using linearization at every time step), 2) the Newton-Schur complement approach (that involves a global linearization) in Alg. 3, and 3) the 1-level nonlinear Schur complement-Newton approach in Alg. 5. Time integration is performed with the Backward Euler scheme with a constant time step. The linearization of the nonlinear problems is performed with a hybrid Picard-Newton technique, where Newton's method is activated when the discrete L_2 -norm of the residual is below 10^2 . (Even though all the algorithms have been stated using the full Newton's method for simplicity, its extension to Picard linearization (and hybrid approaches) is straightforward.) Sequentiality is exploited at the local level in each processor for all the different approaches (level-0 and higher level systems are lower block-triangular; see Eqs. 2 and 5, respectively). The stopping criteria for the sequential and Newton-Schur complement solvers is the reduction of the discrete L_2 -norm of the nonlinear local/global residual below an absolute value of 10^{-8} . For the third approach, the global residual tolerance is fixed to 10^{-8} , while local nonlinear problems and Schur complement solutions converge below 10^{-10} .

The Lotka-Volterra equations are a system of nonlinear ODEs frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other one as a prey. Our unknown functions, namely (u, v) , represent the evolution in time of the number of units of each one of the species considered. The system of nonlinear, first order, differential equations reads

$$\begin{aligned} \frac{du(t)}{dt} &= \alpha u - \beta uv \\ \frac{dv(t)}{dt} &= \delta uv - \gamma v \end{aligned}$$

where $\alpha, \beta, \gamma, \delta$ are positive parameters that model the interaction of the two species. The system is solved for $t \in (0, T]$. Appropriate initial conditions (initial number of preys and predators) must be provided, i.e., $u(0), v(0)$. Out of these equations, and graphically illustrated with an example in Fig. 5, if no interaction between the species is modelled (uncoupled linear equations), the preys grow in number while the predators decrease 5(a). In the second case, the species interaction leads to the frequency plot 5(b).

5.2. Two-level solvers. In this subsection, weak scalability results are presented for the solution of the Lotka-Volterra equations with a two-level method. The local problem size at level-0 is fixed to $\theta = \frac{n_0}{n_1}$. We consider a weak scalability analysis in which we keep fixed the local problem size $\frac{n_0}{n_1}$ and

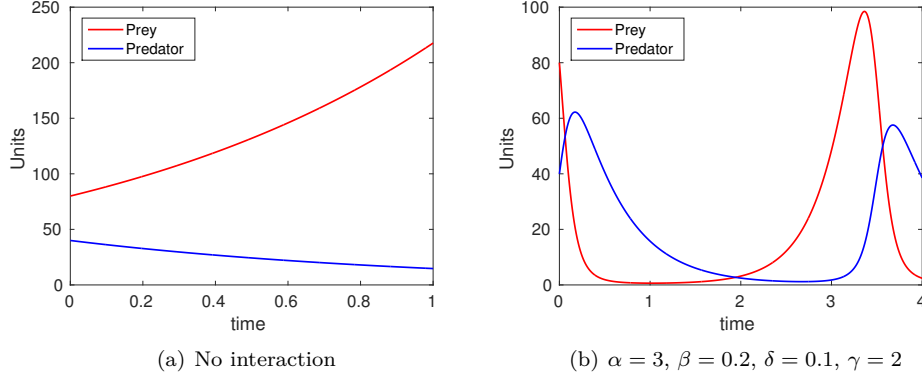
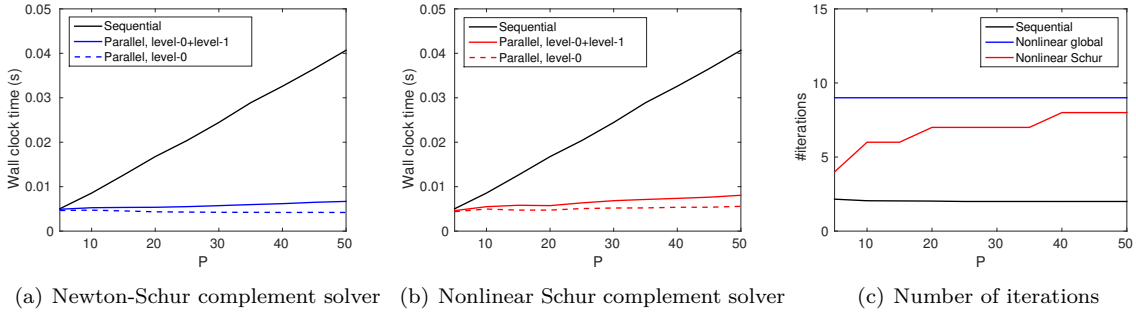


FIGURE 5. Solution to the Lotka-Volterra equations when different scenarios are considered.

increase n_1 , i.e., the number of subdomains (level-1 time elements). Thus, we are increasing the global time steps being used to solve the ODE. The Schur complement at level-1 has size n_1 . We stop the analysis when $n_1 \approx \frac{n_0}{n_1}$, since we would require a three-level algorithm (see Sect. (3.3)).

Figs. 6 and 7 show a comparison between the weak scalability of the sequential solver (1 processor performs sequentially the full computation) and the parallel solvers for three different local problem sizes $\frac{n_0}{n_1}$. In these plots, the parallel solvers computing times are an aggregation of the local solvers (level-0) and the coarse solver (level-1) CPU times. The number of iterations for the sequential approach is an average value for all nonlinear time step problems. For parallel approaches, it shows the number of global nonlinear iterations required to meet the convergence requirements.

FIGURE 6. Weak scalability for local problem size $n_0/n_1 = 50$. The problem is solved with $\alpha = 3, \beta = 0.2, \delta = 0.1, \gamma = 2$ and $t \in (0, 3]$. Initial guess $u(0) = 10, v(0) = 40$.

Out of these plots, we can draw some conclusions. First, parallel approaches reduce from the very beginning the time-to-solution of the simulations. Second, and most important, excellent weak scalability is observed for both parallel solvers; we can solve X times more time steps increasing X times the number of tasks, in approximately the same CPU time.

Regarding the Newton-Schur complement approach, one can observe that the CPU time at level-1 is always less than half the CPU time at level-0. It is due to the fact that we stop the scalability analysis before the coarse space is larger than the level-0 local problems and the fact that two local solvers are required at level-0 (see Alg. 1) per nonlinear iteration. For the nonlinear Schur complement-Newton approach, the number of global iterations to converge is lower in most cases, but it involves nested nonlinear iterations. The Newton-Schur complement approach is slightly faster than the nonlinear Schur complement-Newton one. In any case, it can strongly be affected by the tolerances being chosen

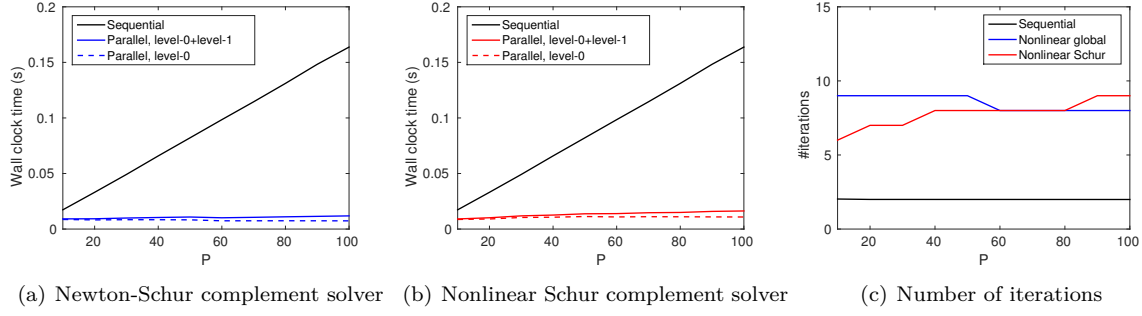


FIGURE 7. Weak scalability for $n_0/n_1 = 100$. The problem is solved with $\alpha = 3$, $\beta = 0.2$, $\delta = 0.1$, $\gamma = 2$ and $t \in (0, 3]$. Initial guess $u(0) = 10$, $v(0) = 40$

in the nonlinear Schur complement-Newton method. Clearly, increasing the load per processor (local problem sizes), the benefit of the parallel solvers compared to the sequential approach becomes more notorious.

Results are presented up to the limit of $n_1 \approx \frac{n_1}{n_0}$, where the coarse problem size grows above the local problem sizes. Aiming to exploit further concurrency, the next section is devoted to show numerical results in a multilevel approach.

5.3. Multilevel solver. The multilevel (three-level) approach is activated when the coarse problem size exceeds the local problem size. The first coarsening leads to a computation with n_1 level-0 local problems of size $\frac{n_0}{n_1}$ and a coarse problem of size n_1 (level-1 partition), such that $n_1 > \frac{n_0}{n_1}$. At this point, another level of coarsening is introduced to exploit further concurrency. We consider a coarsening of the level-1 time partition into n_2 local problems of size $\frac{n_1}{n_2}$ and a coarse problem of size n_2 such that $n_2 < \frac{n_1}{n_2}$ (beyond this limit, a four-level method would be required). In the following plots, the parallel solvers computing times are an aggregation of the level-0 local solver CPU time (n_1 parallel tasks), the level-1 local solver CPU time (n_2 parallel tasks) and the level-2 global solver CPU times (Schur complement sequential solve in one processor).

In Fig. 8, CPU times for the three-level (i.e., $\ell = 2$) Schur complement solve are shown. Out of the plots, it can be observed that local solves computing times for the level-0 and level-1 tasks are of the same order, since local problem sizes is kept constant for both levels. The Schur complement computation (level-2) time grows with the number of processors, as expected. Again, the level-2 the Schur complement CPU time in Fig. 8 does not exceed half the CPU time of level-0 and level-1 local solves CPU time, due to the same reasons commented above, since $n_2 < \frac{n_1}{n_2}$. Beyond this limit, i.e., for $n_2 > \frac{n_1}{n_2}$, another level would be needed.

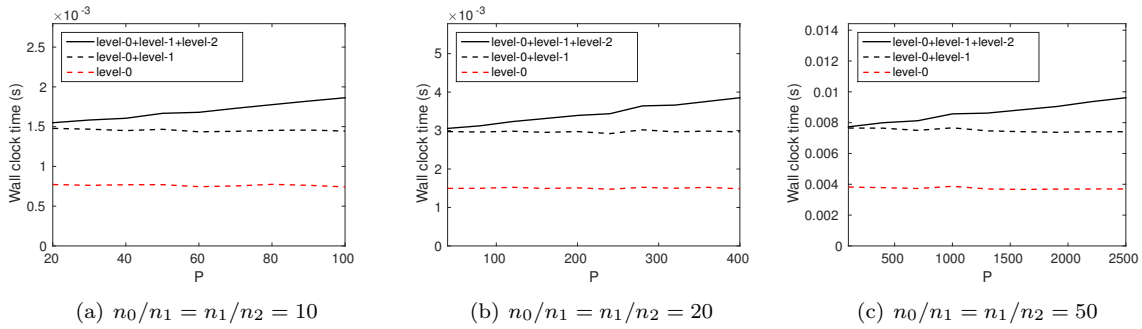


FIGURE 8. Three-level ($\ell = 2$) Schur complement solver with different local sizes.

In Fig. 9, a comparison between the two-level ($\ell = 1$) and the three-level ($\ell = 2$) approaches is presented. In the first part of the plot, results are shown for the two-level approach only, since the size of the level-1 problem is still below the local sizes of the level-0 problems. The three-level technique is activated when the size of the coarse problem is two times the size of the local problems of the finest level, leading to a level-2 partition into two time elements. Out of the plots, the multilevel approach shows much better efficiency than the two-level approach. It is important to note that to include additional levels do not affect nonlinear convergence of Newton-Schur and 1-level nonlinear Schur complement-Newton methods, but it has benefits in the computation of the linear ODE systems. As a result, more levels show better computing times in these approaches.

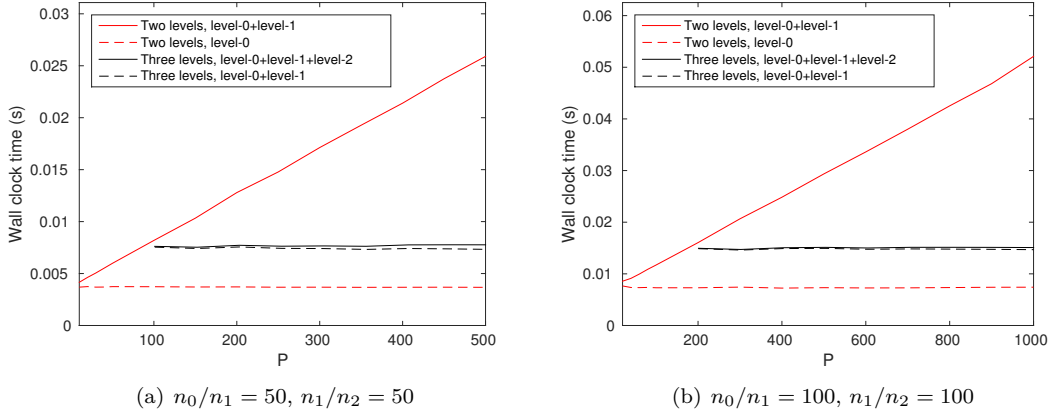


FIGURE 9. Comparison between the two-level ($\ell = 1$) and the three-level ($\ell = 2$) Schur complement solver with different local sizes. Times for local solvers in the level-0, local solvers in the level-1 and the Schur complement solve in level-2.

In Fig. 10, an adaptive coarsening at level-2 is presented. Since $n_2 < \frac{n_1}{n_2}$ in the plot, we can reduce the coarsening from level-1 to level-2 to better balance the problems at these levels. It implies to enforce that $n_2 = \frac{n_1}{n_2}$. As expected, this choice is more efficient compared to the fixed size approach, when the size of the level-2 problem is below level-1 local problem sizes.

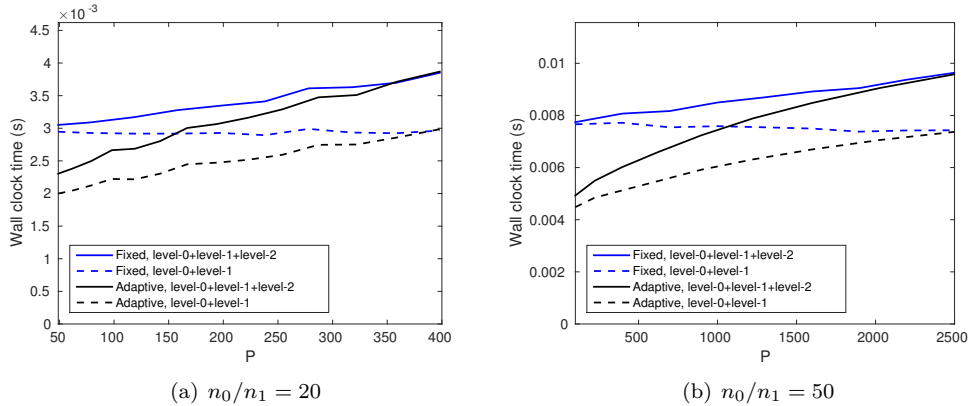


FIGURE 10. Comparison between the fixed coarsening and the adaptive coarsening between level-1 and level-2 for different level-0 local problem sizes. The adaptive coarsening uses $n_2 = n_1/n_2 = \sqrt{n_1}$.

6. CONCLUSIONS

In this work, we have considered a parallel-in-time multilevel direct method and two iterative parallel-in-time nonlinear solvers for the numerical approximation of ODEs using parallel computations. The time-parallel direct method computes explicitly multi-level Schur complements. It allows for arbitrary high levels of concurrency and has very good theoretical speed-up ratios compared to traditional parareal methods. The method can be considered a parareal method with an automatic definition of the coarse solver. However, such definition makes the method to converge in one iteration, i.e., it is a direct method, and thus does not suffer from the poor parallel efficiency of parareal schemes. The proposed scheme can be applied to θ -methods, DG methods, Runge-Kutta methods, and BDF methods. Next, we consider nonlinear ODEs, and propose two different strategies. First, we consider a Newton method over the global nonlinear ODE, using the multilevel Schur complement solver at every nonlinear iteration. Second, we state the global nonlinear problem in terms of the nonlinear Schur complement (at an arbitrary level), and perform nonlinear iterations over it. Such approach leads to nested nonlinear multilevel iterations.

A detailed set of numerical experiments has been considered. Out of these results, the proposed methodology is observed to exhibit excellent scalability properties. The methods are weakly scalable in time, i.e., increasing X times the number of MPI tasks one can solve X times more time steps, in approximately the same amount of time, which is a key property to reduce time-to-solution in ODE simulations with heavy time stepping.

We refer to [5] for the combination of these ideas with space-parallel highly scalable BDDC implementations [2–4] to design space-time preconditioners for transient PDEs. The nonlinear algorithms in [5] only consider the Newton-Schur complement solver. Future extensions of this work is the application of the level- k nonlinear Schur complement scheme in space, leading to new nonlinear space-time preconditioners.

REFERENCES

- [1] S. BADIA, A. F. MARTÍN, AND J. PRINCIPE, *Implementation and Scalability Analysis of Balancing Domain Decomposition Methods*, Archives of Computational Methods in Engineering, 20 (2013), pp. 239–262.
- [2] ———, *A Highly Scalable Parallel Implementation of Balancing Domain Decomposition by Constraints*, SIAM Journal on Scientific Computing, 36 (2014), pp. C190–C218.
- [3] ———, *On the scalability of inexact balancing domain decomposition by constraints with overlapped coarse/fine corrections*, Parallel Computing, 50 (2015), pp. 1–24.
- [4] ———, *Multilevel Balancing Domain Decomposition at Extreme Scales*, SIAM Journal on Scientific Computing, (2016), pp. C22–C52.
- [5] S. BADIA AND M. OLM, *Space-time balancing domain decomposition*, In press, SIAM Journal on Scientific Computing. Also available at arXiv:1701.03477 [cs], (2017).
- [6] A. BELLEN AND M. ZENNARO, *Parallel algorithms for initial-value problems for difference and differential equations*, Journal of Computational and Applied Mathematics, 25 (1989), pp. 341–350.
- [7] X.-C. CAI AND D. E. KEYES, *Nonlinearly preconditioned inexact Newton algorithms*, SIAM Journal on Scientific Computing, 24 (2002), pp. 183–200.
- [8] A. J. CHRISTLIEB, C. B. MACDONALD, AND B. W. ONG, *Parallel high-order integrators*, SIAM J. Sci. Comput., 32 (2010), pp. 818–835.
- [9] C. R. DOHRMANN, *A preconditioner for substructuring based on constrained energy minimization*, SIAM Journal on Scientific Computing, 25 (2003), pp. 246–258.
- [10] M. EMMETT AND M. J. MINION, *Toward an efficient parallel in time method for partial differential equations*, Comm. App. Math. and Comp. Sci., 7 (2012), pp. 105–132.
- [11] R. FALGOUT, S. FRIEDHOFF, T. KOLEV, S. MACLACHLAN, AND J. SCHRODER, *Parallel time integration with multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C635–C661.

- [12] M. J. GANDER, *50 years of time parallel time integration*, in Multiple shooting and time domain decomposition, Springer, 2015.
- [13] M. J. GANDER AND S. GÜTTEL, *ParaExp: A parallel integrator for linear initial-value problems*, SIAM J. Sci. Comput., 35 (2013), pp. C123–C142.
- [14] A. KLAWONN, M. LANSER, AND O. RHEINBACH, *Nonlinear FETI-DP and BDDC Methods*, SIAM Journal on Scientific Computing, 36 (2014), pp. A737–A765.
- [15] J. LIONS, Y. MADAY, AND A. TURINICI, *A parareal in time discretization of PDEs*, Acad. Sci. Paris, 332 (2001), pp. 661–668.
- [16] J. MANDEL, B. SOUSEDÍK, AND C. DOHRMANN, *Multispace and multilevel BDDC*, Computing, 83 (2008), pp. 55–85.
- [17] X. TU, *Three-level BDDC in three dimensions*, SIAM Journal on Scientific Computing, 29 (2007), pp. 1759–1780.